Improving the Network Energy Efficiency in MapReduce Systems*

Lin Wang*‡, Fa Zhang*, Zhiyong Liu*§
*Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
‡University of Chinese Academy of Sciences, Beijing, China
§Chinese State Key Lab for Computer Architecture, ICT, CAS, Beijing, China

***Abstract*—Apart from servers, the energy consumed by enormous amount of network devices in data centers also emerges as a big problem. Existing work on energy-efficient data center networking primarily focuses on traffic engineering to consolidate flows and shut down unused devices, not considering another important factor, virtual machine assignment, which has been shown to have a big influence on traffic engineering. Moreover, the lack of information about upper layer applications leads to misunderstand the traffic patterns of the network. This may result in poor effectiveness in the traffic-based optimization in practice. In this paper, we aim to achieve better network energy efficiency in MapReduce systems by combining virtual machine assignment and traffic engineering. By exploiting the characteristics of MapReduce applications, we provide a unified model to describe this problem. Due to its NP-hardness, a general framework is proposed to solve it, where virtual machines are first clustered and then different virtual machine assignments are generated greedily while a local search procedure is used to improve them. The local search procedure depends on the results of an energy-efficient routing provided by GEERA. GEERA is an approximate algorithm designed to select routing paths for flows. Experimental results confirm the efficiency of GEERA, as well as the overall framework. By using this framework, up to 20% more energy savings can be achieved compared with sole traffic engineering solutions.**

***Index Terms*—Energy efficiency, Data center networks, MapReduce, Virtual machine assignment, Traffic engineering**

I. Introduction

Cloud computing has been widely adopted in many companies such as Google, Microsoft and Yahoo!. In order to efficiently process large quantities of data produced by online services such as search engine and social networks, MapReduce [1] has been popularized by Google as a specialized programming paradigm to take advantage of cloud resources, especially for the *Big Data* processing.

Data centers are integrated facilities to house computer systems for cloud computing. To support high-level applications, a huge number of connected servers have to be placed in a data center. As a result, the huge energy consumption in data centers has become a crucial problem [2]. To handle this situation, many solutions have been proposed to reduce the power consumed by servers, including dynamic voltage frequency scaling (DVFS) and virtual machine migration and consolidation. Hereafter, the energy consumed by enormous amount of network elements for connecting these servers then emerges as a substantial issue. Abts et al. [3] showed that in a typical data center from Google, the network power is around a fraction of 20% to the total power with 100% utilized servers, but it increases to 50% when the utilization of servers decreases to 15%, which is quite realistic in production data centers. Therefore, improving the energy efficiency of the network also becomes a primary concern.

There have been a significant amount of work on providing energy-efficient networking in data centers (e.g. [4], [5], [6], [7], [8], [9], [3], [10]). The most straightforward way to save energy in a data center network is by consolidating traffic flows and shutting down unused network devices. This is inspired by an observation that data center networks are usually designed with big connectivity redundancy to provide fault tolerance and to handle peak traffic, but the network load in a data center keeps at a quite low level in most of the time. Although a considerable amount of energy can be saved, there exist some limitations in applying this method. In general, the traffic in a data center network jitters very frequently, though a long-term pattern may be found [11]. To achieve real-time adaption of the energy saving solution, traffic prediction is necessary to be involved, which is not feasible or precise enough in most data center networks, leading to inefficient saving results.

In order to overcome the above obstacles, we propose a new energy saving model for data center networks. The basis of this model is formed by two aspects: application characteristics and virtual machine assignment. In particular, we focus on MapReduce systems, since MapReduce jobs are representative in cloud data centers. Nevertheless, the proposed model is not exclusive to MapReduce systems and can also be applied in normal data centers. In previous work, energy saving strategies were mostly conducted without any knowledge of the running applications. As a result, obtaining essential information of the network traffic patterns would be quite hard, resulting in the need of optimizing the system every time there is a considerable change on the traffic condition, and therefore, leading to unstable efficiency in practice. We believe that it is critical to consider the application-level traffic characteristics from upper layer when optimizing the performance of lower layer infrastructures. Another observation is that unlike in traditional networks, the endpoints of traffic flows in a data center network can be determined artificially by virtual

*This research was supported in part by the National Natural Science Foundation of China grant 61020106002, 61161160566 and 61202059, and the Comunidad de Madrid grant S2009TIC-1692, Spanish MICINN grant TEC2011-29688-C02-01.

978-1-4673-5775-3/13/$31.00 ©2013 IEEE

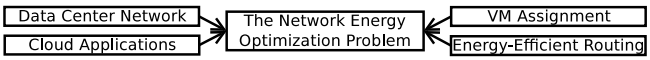


Fig. 1. An overview of the model

machine assignment, leading to a remarkable influence in the traffic patterns of the network and, thus, conditioning the traffic engineering. In principle, an optimization combining both traffic engineering and virtual machine assignment will provide substantial improvements to the energy efficiency in data center networks.

In a MapReduce system, many jobs are processed simultaneously. Each job is divided into small tasks and distributed to many virtual machines. A MapReduce job is also divided into three different phases: Map, Shuffle and Reduce. During Shuffle phase all the virtual machines implied in one job will exchange their Map results, generating a large amount of traffic. This phase is known to cause some important problems as peaks in the energy consumption or in the congestion of the network. Since jobs are not necessarily synchronized and Shuffle phases may occur at different instants of time, it is then clear that the virtual machine assignment will have a significant influence on the network traffic pattern, allowing us to reduce substantially both congestion and energy consumption by combining virtual machine assignment and traffic engineering. At the same time, the optimization results will be independent from traffic variations.

In this paper, we combine virtual machine assignment and traffic engineering to improve the network energy efficiency in MapReduce systems. To the best of our knowledge, this has not been addressed in-depth before. We propose a unified model that models the data center, the application and the network energy optimization problem (NEOP). By characterizing the traffic patterns of applications in an explicit way, the number of optimizations to be performed in the system will be reduced as they will be independent from the traffic variations. We then develop an efficient framework to solve NEOP, where virtual machine assignment will depend on traffic engineering results. The experimental results on different topologies show that considering both virtual machine assignment and traffic engineering together can bring a substantial improvement to the energy efficiency of the network.

The reminder of this paper is organized as follows. Section II formally describes the problem and provides a unified model. Section III presents a general framework and our main algorithms to solve the problem. Section IV demonstrates the experimental results, where our proposed algorithms are evaluated by simulations. Section V concludes the paper.

II. The Network Energy Optimization Model

We define the problem and provide a formal description of our unified model in this section. We only focus on the network energy consumption but still our work can be incorporated in energy-saving strategies for physical servers. An overview of the model can be found in Fig. 1.

TABLE I
Power Rating Profiles of Some Typical Commodity Switches (Unit: Watths)

Product	Idle or Nominal	Max
Cisco Nexus 3548	152	265
Cisco Nexus 5548P	390	600
HP 5900AF-48XG	200	260
HP 5920AF-24XG	343	366
Juniper QFX 3600	255	345

A. The Data Center

We model a data center as a homogeneous system where a set of identical physical servers are connected by a particularly designed network. Suppose there are a set of n servers denoted by $S = \{s_1, s_2, ..., s_n\}$. In order to make use of hardware resources more efficiently, jobs are processed by virtual machines hosted by these physical servers. We assume identical virtual machines as it is the normal case in MapReduce systems. Each physical server can host at most m virtual machines due to resource limitation. All the physical servers are connected by a network $G = (V, E)$ where V is the set of nodes (servers S and switches W, $V = S \cup W$) and E is the set of links. To take the place from the traditional tree structure, recently many sophisticated network topologies have been proposed to provide more scalable and reliable networking for data centers, including FatTree [12], BCube [13] and DCell [14]. For link $e \in E$, we assume the traffic load on it is y_e. Then, for node $w \in W$, the traffic load on w is the summation of the total amount of flows going through it, which can be represented as

$$x_w = \frac{1}{2} \sum_{w \text{ is incident to } e} y_e, \tag{1}$$

where the factor $\frac{1}{2}$ is used to eliminate the double counting of each flow.

In order to achieve energy proportionality, there have been a lot of energy saving strategies proposed for single network devices. Among them, it is believed that the most efficient one is to power off the device when it is idle. Since the chassis of a network element takes a big fraction of the total energy consumption, switching off a device will result in the most substantial saving. In order to examine this result, we extract the power profiles of some typical commodity switches, and show them in Table I. It can be noticed that even though the idle power is dominant, the operational power is also considerable and cannot be ignored. For instance, the operational power takes nearly 40% of the total consumption in both Cisco and Juniper data center switches. As a result, a better model is needed to characterize the power consumption of network elements.

We use the speed scaling model in which we assume each node $w \in W$ has the speed scaling capability, characterized by an energy curve $f_w(x_w)$, which is how w consumes energy as a function of its transmission speed x_w. Particularly, we use

a function $f_w(x_w)$ in the following form.

$$f_w(x_w) = \begin{cases} 0 & \text{for } x_w = 0 \\ \sigma_w + \mu_w x_w^{\alpha_w} & \text{for } x_w > 0 \end{cases}, \quad (2)$$

where σ_w represents a fixed amount of energy needed to turn on a device, and μ_w and α_w are parameters associated with devices. Since the energy consumption of a network device grows superadditively with its load, α_w is usually larger than 1 and has been shown to be in $(1, 3]$ [15]. The reason why we choose this power function is that this function can take both power down and rate adaptation into consideration in the meantime, which is most realistic to the real power consumption statistics shown above.

The cost of a network is defined as the total power consumption of all its network devices. Since a data center is a homogeneous system as assumed, we consider the case that all the switches consume energy in the same way. That is, for every node in W, a uniform power function $f(\cdot)$ is given to express its corresponding power consumption. The total power rate of the network then can be formulated as

$$P = \sum_{w \in W} f(x_w) = \sum_{\{w \in W | x_w > 0\}} (\sigma + \mu x_w^{\alpha}). \quad (3)$$

B. The Application

We consider the MapReduce-style applications as we focus on MapReduce systems. A typical MapReduce application consists of three main phases: Map, Shuffle and Reduce. However, the network utilization during the three phases is quite different. In the Shuffle phase, the partial computation results of the Mappers are transfered to the virtual machines performing the Reduce operation, resulting in a massive amount of traffic flows on the network. But in the other two phases, the computing resources are intensively used and the traffic on the network stays very small.

Due to the three-phase pattern MapReduce possesses, we conjecture that the communication patterns of typical cloud applications should be able to be classified into several categories. Actually this has already been confirmed by Xie *et al.* [16]. In that work, the authors profiled the network patterns of several representative jobs of MapReduce, including Sort, Word Count, Hive Join and Hive Aggregation which most commonly appear in data centers. They observed that all the jobs generate considerable traffic during only 30%-60% of the entire execution. Moreover, the traffic patterns of these cloud applications can mainly be classified into three major categories: single peak, repeated fixed-width peaks and varying height and width peaks. They also provided exhaustive profiling techniques to obtain the traffic patterns of jobs.

Assume we are given a set J of MapReduce jobs to be processed simultaneously. Each job $j \in J$ is partitioned into η_j tasks that need to be accommodated by η_j virtual machines. Suppose we have numbered all the virtual machines from 1 to $\sum_{j \in J} \eta_j$. For the ease of modeling, we consider a MapReduce job as a set of transfers R_j. A *transfer* is defined as a Shuffle phase of a MapReduce job in which the Map virtual machines belongs to this job send their computing results to the corresponding Reduce virtual machines. We consider only transfers because the traffic generated by a job is very significant only in transfers. Each transfer $r \in R_j$ is accompanied by a time interval $[t_r^s, t_r^t]$ during which this transfer is being proceeded, and a traffic matrix T_r indicating the traffic between each pair of the virtual machines belonging to the job during this transfer. In order to avoid packet reordering, we adopt integral routing, where each flow can only be routed through a single path from its source and destination.

C. The Network Energy Optimization Problem

We define now the network energy optimization problem (NEOP) for data centers in this subsection. Denote $t^s = \min\{t_r^s \mid r \in \cup_{j \in J} R_j\}$ and $t^t = \max\{t_r^t \mid r \in \cup_{j \in J} R_j\}$. Without loss of generality, we concentrate only on time period $[t^s, t^t]$. The total energy consumed by all network elements after processing all jobs then can be represented as

$$E = \int_{t^s}^{t^t} P(t) = \int_{t^s}^{t^t} \left(\sum_{w \in W} f(x_w(t)) \right) dt \quad (4)$$

where $P(t)$ is the power rate of the whole network and $x_w(t)$ is the load of node w at time t. Since virtual machine migration is usually time-consuming and brings big traffic overhead, we assume here that once the virtual machines for a MapReduce job have been assigned to physical servers, they will never be migrated until the job completion. Our goal is to assign all the virtual machines to physical servers in such a way that when we choose the appropriate routing paths for the flows in transfers, the total energy consumption E of the network is minimized.

Actually this optimization procedure can be divided into two closely related stages: virtual machine assignment and traffic engineering. Virtual machine assignment will bring substantial influence to the traffic conditions on the network as well as the network power consumption, and will also depend on the traffic engineering results of reducing the network power consumption. Assume we are given an algorithm $\mathcal{A}$ for solving the network power minimization problem for traffic engineering, returning the total power rate of the network. The virtual machine assignment problem then can be formulated as below.

$$\begin{aligned} & (P_1) \quad \min \ \int_{t^s}^{t^t} \mathcal{A}(D(t))dt \\ \text{subject to} \quad & \\ & \textstyle\sum_x u_{xy} \le m && \forall y \\ & \textstyle\sum_y u_{xy} = 1 && \forall x \\ & u_{xy} \in \{0, 1\} && \forall x, y \end{aligned}$$

where u_{xy} indicates whether the x-th virtual machine ($x \in [1, \sum_{j \in J} \eta_j]$) is assigned to the y-th server ($y \in [1, n]$). $D(t)$ is the set of traffic demands which need to be routed at time t. Each traffic demand in $D(t)$ is indicated by a triple consisting of a source, a destination and a flow amount. Once the virtual machines have been allocated, $D(t)$ can be obtained from the traffic matrices of transfers.

We then dispose of the energy-efficient routing problem which algorithm $\mathcal{A}$ aims to solve. After obtaining the traffic demands $D(t)$, this problem can be represented as follows: given a network $G = (V, E)$ with a cost function $f(\cdot)$ and a set of traffic demands $D(t)$, the goal is to unsplittablely route all demands, such that the total cost of the network $\sum_w f(x_w)$ is minimized, where x_w is the total load on $w \in W$. $W \subset V$ is the set of nodes with degrees larger than 1. (These nodes correspond to the switches in a data center network.) Formally, it can be formulated as the following integer program.

$$
\begin{aligned}
& (P_2) \quad \min \ \textstyle\sum_w f(x_w) \\
\text{subject to} \\
& x_w = \tfrac{1}{2} \textstyle\sum_{w \text{ is incident to } e} y_e && \forall e \\
& y_e = \textstyle\sum_{i \in [1, |D(t)|]} y_{ie} && \forall e \\
& x_w \leq Z \\
& y_{i,e} \in \{0, 1\} && \forall i, e \\
& y_{i,e} : \text{ flow conservation}
\end{aligned}
$$

where y_{ie} is an indicator variable to show whether demand i $(1 \leq i \leq |D(t)|)$ goes through edge e. Z is the capacity of the switch. Flow conservation means only a source (sink) node can generate (absorb) flows, while for other nodes the ingress traffic equals the egress traffic. y_e is the total load carried by link e and x_w is the total traffic going through node w.

III. Methodology

In this section, we present a general framework and some algorithms in this framework for solving the problem. It can be proved that solving NEOP is NP-hard by using a reduction from the Quadratic Assignment Problem (QAP) which has been shown to be NP-hard [17]. With the fact that we cannot obtain the optimal solution for this problem in polynomial time, we devise a general framework with four components, solving the problem heuristically by dividing the big problem into small pieces. We now introduce the components of the framework one by one.

A. Virtual Machine Consolidation

We first consider how to assign virtual machines to single servers. As we assumed, each physical server can hold at most m virtual machines at the same time. One observation we can make is that once we put two virtual machines together in the same server, the traffic between the two virtual machines (if exists) will not go through the network any more. It then can be observed that if we consolidate virtual machines with large communication traffic between each pair of them, the total traffic on the network will be reduced. As a result, the total power consumption of the network can be accordingly cut down.

The problem of consolidating virtual machines with large inter-communication traffic can be solved by using the algorithm for the minimum k-cut problem. The minimum k-cut problem aims at finding a set of edges whose removal would partition a graph to k connected components such that the total weight of all the removed edges is minimized. If k is part of the input, the minimum k-cut problem is NP-complete. But there exists many good approximations. Our virtual machine consolidation problem can be easily transformed to the minimum k-cut problem by building a complete graph with all the virtual machines as nodes and the traffic between each pair of virtual machines as the weight between them. The problem is to find a collection of edges such that the summation of the weights on these edges is minimized, and also each partition generated by the removal of these edges will have a specified size of m. As have been studied, the virtual machine consolidation problem is a special case of the minimum k-cut problem and is approximable within a factor of 3 for any fixed k.

After this consolidation, the model we proposed in Section II has been simplified where each physical machine will accommodate a set of specified virtual machines which can be regarded as a super virtual machine. We build a new traffic matrix for each job with respect to super virtual machines and each unit in this matrix is the traffic between any pair of two super virtual machines.

B. Virtual Machine Assignment

The assignment of virtual machines can be accomplished by a heuristic based on local search. This heuristic works repeatedly with independent attempts. Each attempt consists of two phases, initiation and local search. We denote $\pi \in \Pi_n$ as a permutation of all the super virtual machines and cost $E(\pi)$ as the total energy consumption using permutation p as the order to assign all the super virtual machines to physical servers. $E(\pi)$ can be obtained by solving an energy-efficient routing problem on the network, which will be discussed later.

In the initiation phase, initial permutations are generated. This is done by first choosing two virtual machines from two different jobs at random and assigning them to servers. Afterwards, we assign the other virtual machines one by one. Let $\Omega = \{(x_1, y_1), ..., (x_q, y_q)\}$ denote the first q allocations made. Then the cost of assigning a new virtual machine to a server is calculated as the increase in the total cost caused by this new allocation. To make the $(q+1)$-th allocation, select an allocation at random from among the feasible allocations with smallest costs and add the allocation to Ω.

Once an initial solution is constructed, local search is applied to it to try to minimize its cost $E(\pi)$. For each pair of allocations in the current solution, check if the swap of the two allocations reduces the cost of the assignment. If so, make the swap. When no swap decreases the cost, a local optimal is obtained. The algorithm terminates when a determined number of iterations have been finished. The solution with the lowest cost is output as the final solution.

Although the local search based heuristic cannot guarantee the optimal assignment for virtual machines, we will show that it is efficient enough in practice. Due to the virtual machine consolidation we carried out before this heuristic, the complexity of this local search has been decreased dramatically and becomes reasonable. Meanwhile, the implementability is the most competitive factor compared with other complicated

approximation algorithms.

C. Transfer Management

Now we consider how to obtain the value of $E(\pi)$. Recall that $E(\pi)$ is the integral of the network power rate over time period $[t^s, t^t]$ and the power rate of the network at a time can be obtained from the traffic matrices of the active transfers at that time. As we have consolidated virtual machines to super virtual machines, accordingly, the traffic matrices of transfers have to be transformed to be with super virtual machines. To computer the power rate of the network at each time in $[t^s, t^t]$, we need to know the number of active transfers at that time, from which the network traffic condition can be determined. It can be observed that there is no need to calculate the power rate of the network at every time, because the network traffic maintains the same if the set of active transfers does not change. By using the time interval attached to each transfer, it is easy to obtain all the network-stable periods in which the same set of transfers is active. Then, for each network-stable period, we compute the network traffic matrix from the traffic matrices of the corresponding active transfers and obtain the network power rate by applying an energy-efficient routing procedure which is algorithm $\mathcal{A}$ as we have discussed in Section II.

D. Energy-Efficient Routing

We dispose of the problem that algorithm $\mathcal{A}$ aims to solve in this subsection. The goal of the problem is to find appropriate routes for flows such that the total power rate of the network is minimized. Actually, this problem is an extended case of a network design problem with convex node costs. For the edge-weighted version, this network design problem has been proved to be NP-hard [15]. The edge-weighted version can be easily reduced to the node-weighted version by introducing an extra node on each edge and moving the cost function on edges to the newly added nodes. Thus, we can conclude that the energy-efficient routing problem with convex node costs is also NP-hard. However, for the non-subadditive and non-superadditive cost function, the energy-efficient routing problem is much harder than these problems [18]. In fact the difficulty of our problem resides in how to balance the startup cost σ and the traffic cost μx^α. This happens because the startup cost encourages aggregating flows to reduce the number of active links, while the convex nature of the traffic cost encourages load balancing traffic.

As in each iteration the energy-efficient routing problem needs to be solved, it is necessary to minimize the time for solving it. For this reason, we provide a heuristic, *GEERA*, which can balance the two parts of the cost and give efficient results. The biggest advantage, comparing *GEERA* to some linear programming (LP) based algorithms, is that it solves the problem very fast while providing nearly optimal solutions. Before describing the algorithm, we first provide some terminology.

Definition 1. *The power rate of a network element is defined as $f(x)/x$ $(x > 0)$, which is the average power consumed per traffic load. The optimal traffic load x^* for devices is obtained when the power rate $f(x)/x$ is minimized. That is $x^* = \sqrt[\alpha]{\frac{\sigma}{(\alpha-1)\mu}}$.*

Definition 2. *The overload penalty is an extra cost to a node whose load exceeds x^*. We define this penalty to be equal to the startup cost σ.*

Algorithm 1 GEERA

Input: $G = (V, E)$ (undirected graph, $W \subset V$), $D = \{(s, t, d_i) \mid i \in [1, |D|]\}$ (set of traffic demands), $f(\cdot)$ (cost function on nodes)

```
Sort all the demands in D in descending order
C_w ← 0, I_w ← 0 (∀w ∈ W)
W_e ← 0 (∀e ∈ E)
for demand i ∈ |D| do
    for each w ∈ W do
        if C_w + d_i > x* then
            I_w = f(C_w + d_i) − f(C_w) + σ
        else
            I_w = f(C_w + d_i) − f(C_w)
        end if
    end for
    for each e ∈ E do
        Set the weight of edge e: W_e = ½(I_{w_1} + I_{w_2}), where e = (w_1, w_2)
    end for
    Find the shortest path H_i for demand i by using the weight W_e on each edge e and route demand i via H_i
    C_w ← C_w + d_i if w ∈ H_i
end for
Return paths for all demands H = {H_i | i ∈ [1, |D|]} and total cost P = Σ_w f(C_w)
```

The pseudo-code of *GEERA* is listed in Algorithm 1. At the beginning we sort all the traffic demands in descending order. This is because routing bigger demands first can reduce the uneven distribution of traffic due to the greedy strategy. Then, all demands will be routed one by one. For each demand, we calculate the increment on the cost of each node by assuming that this demand will be routed through this node. Suppose the current load on each node is C_w and the traffic of the current demand is d_i. For a node w, we consider two different cases for the increment on the cost: if $C_w + d_i \leq x^*$, we route this demand through w and set the increment on the cost to be $f(C_w + d_i) - f(C_w)$; Otherwise, we add an overload penalty σ to the increment on the cost. Then, the priority of our algorithm will be to select paths with as less overloaded nodes as possible. This is because the most energy-efficient situation is when all network elements carry a traffic load equaling to x^*. Thus, by using the increment on cost of each node as its weight, the shortest path between the source and destination in the node-weighted graph is selected as the routing path for this demand. This node-weighted shortest path problem is solved by transforming it to a normal edge-weighted shortest path

problem, by moving the weights on nodes to edges, as shown in line 13. The weight for an edge is regarded as half of the total weight of its end nodes. Observe that for any intermediate node, each flow that goes into a node will go out. A flow going through an intermediate node will take one half of the node weight from its ingress edge and the other half from its egress edge. However, for a source or a destination node, only half of the node weight is carried by its egress edges or ingress edges. This is the same for all the candidate paths and thus does not really affect the final result. The edge-weighted $s-t$ shortest path can be obtained by the Dijkstra's algorithm. When terminating, the algorithm returns the set of paths for routing all demands and the corresponding total cost.

E. Dynamic Job Arrival and Departure

The proposed algorithm can be adapted to online cases. Recall that the key of the problem we want to solve is to ensure that the power rate of the network is as minimized as possible. We discuss how to adapt our algorithm in online scenarios in two cases: job arrival and job departure.

Job Arrival. When a job arrives, the configuration of the system (virtual machine assignment and routing scheme) needs to be updated. As we have assumed, when a virtual machine has been assigned to a designated server, it will never be migrated to other servers until the task in it has been finished. As a result, the problem now is how to assign the new virtual machines for the arriving jobs and how to obtain new routing schemes. We apply the virtual machine consolidation and assignment in our proposed framework to the newly arrived jobs to obtain the virtual machine assignment. Then, the set of transfers in the network is updated by adding the transfers belonging to these new jobs to the previous set. The energy-efficient algorithm can then be applied to the set of transfers, generating energy-efficient routing schemes for the system.

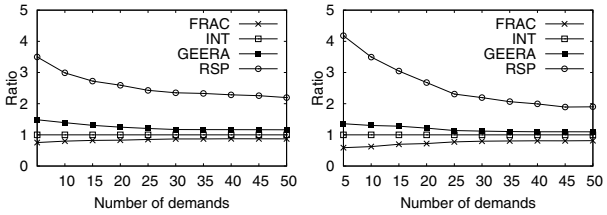
Job Departure. When a job departs, the only thing we need to do is to update the routing schemes. After removing the departed jobs, a new set of transfers can be obtained. Then, by applying the energy-efficient routing algorithm to these transfers, new energy-efficient routing schemes are generated.

IV. EVALUATIONS

We have conducted experiments to validate our proposed framework and algorithms. Particularly, we verify the performance of the energy-efficient routing algorithm GEERA as well as the overall framework.

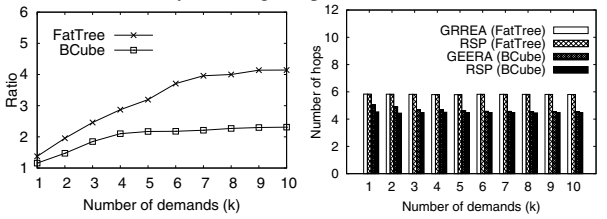
A. Experimental Settings

Most of our experiments are conducted on two kinds of topology: FatTree and BCube which are representative for switch-centric and server-centric topology respectively. To verify the efficiency of GEERA, we randomly generate traffic demands and inject them into the network. The endpoints of these demands are chosen uniformly at random, while the loads of demands follow a normal distribution. For the framework efficiency testing, we generate jobs such that the number of virtual machines for each job is generated uniformly at random, and the total number of virtual machines equals the number of servers. This is done by using Markov Chain Monte Carlo (MCMC) method. The traffic among each pair of the virtual machines for each job is generated randomly following the same normal distribution as before. The start and end times of transfers are selected uniformly from a determined period a time.



(a) Fat-tree (b) BCube

Fig. 2. Comparison of the energy consumption between RSP, GEERA, the integral optimal solution and the fractional optimal solution when $\sigma = 0$. All values are normalized by the integral optimal solution.



(a) (b)

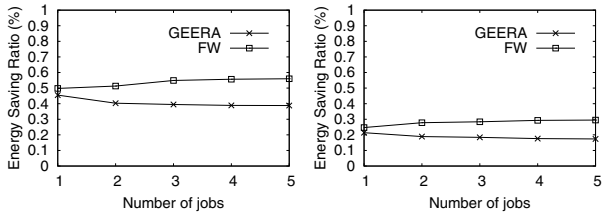
Fig. 3. a) The ratio of the energy consumption of GEERA and RSP when $\sigma > 0$ with different topologies; b) The average number of hops for routing a demand using GEERA and RSP with different topologies

B. Efficiency of GEERA

We compare GEERA with the optimal solution and another load-balancing based heuristic: randomized shortest path (RSP) routing. RSP aims at choosing a path uniformly at random from all candidate shortest paths for each demand. As the cost function $f(\cdot)$ is neither subadditive nor superadditive, we cannot obtain the optimal solution by convex programming. So we first test the case where $\sigma = 0$, and afterwards, we compare GEERA with RSP using a function where $\sigma > 0$.

Case 1: $\sigma = 0$. The experiments are conducted on a 4-ary FatTree and BCube(2,2) topologies. We compare the total cost of GEERA with RSP under different number of demands, as well as to the integral optimal solution and to the fractional optimal solution obtained with CPLEX [19]. All the results are normalized by the integral optimal solution and are shown in Fig. 2. It can be observed that GEERA always obtains nearly optimal solutions, while the costs of RSP are at least two times of the optimal.

Case 2: $\sigma > 0$. We use 8-ary FatTree and BCube(8,2) topologies. We record the total energy consumption while performing GEERA and RSP under different number of demands. The experimental results are presented in Fig. 3(a). We can see that for FatTree, GEERA can achieve significant energy savings when compared with RSP. This happens because GEERA can balance the startup cost σ and the traffic cost μx^{α} better than RSP. For BCube, RSP has better results than with



(a) FatTree (b) BCube

Fig. 4. Energy savings achieved by GEERA and the overall framework.

FatTree, because there are less equal-length paths in between any two servers in BCube. Even though, GEERA can save about a half of the energy compared with RSP. Fig. 3(b) shows the average hops for routing a demand when using the different routing methods. We also notice that the overhead to the average hops for routing flows in GEERA is quite small, consequently the influence in the network latency is negligible.

C. Efficiency of the Overall Framework

We also conduct quantitative comparison between our overall framework and the baseline approach. The baseline we choose is the result obtained by compacting virtual machines for all jobs one by one and routing with RSP, which is a common practice. Since the general model with multiple virtual machines on each server is transformed to the single virtual machine case (as explained in the previous section), here we only focus on the single virtual machine case. Using the baseline E_{BL}, the energy saving ratio is calculated as

$$r = \frac{E_{BL} - E}{E_{BL}} \times 100\%, \tag{5}$$

where E is the energy consumption obtained by GEERA or by the overall framework (FW). The experimental results are illustrated in Fig. 4. We can observe that in this case, GEERA can achieve energy savings of 40% and 20% on FatTree and BCube respectively. The reason why these ratios are smaller than in the previous tests is that the traffic here is less random. When the number of jobs increases, the number of traffic demands decreases, bringing a reduction on the savings achieved by GEERA. On the other hand, by using the virtual machine assignment, the total energy saving increases with the growth of the number of jobs. This demonstrates that the joint optimization of the virtual machine assignment and the traffic engineering brings a significant benefit for improving the energy efficiency of data center networks. In general, our framework can achieve an average energy savings of 60% on fat-tree, and 30% on BCube. Note that these savings are gained by the scheduling and routing algorithms, except for the ones achieved by shutting down network elements.

V. Conclusions

In this paper we explored how to improve the network energy efficiency in MapReduce systems by combining virtual machine assignment and traffic engineering. Unlike in previous works, we propose to make use of the characteristics of upper layer applications, which we believe to have a big influence on the energy optimization for data center networks. By exploiting the characteristics of MapReduce applications, we model the problem with a unified model and propose a general framework to solve it. In this framework, virtual machines are first consolidated into clusters, each of which can be assigned to a single server. Then, independent attempts are carried out to find the best virtual machine assignment. In each attempt, an initial assignment is first greedily generated and then a local search procedure is applied to improve it. The local search procedure is based on the results from an energy-efficient routing problem that is efficiently solved by a greedy heuristic called GEERA. The simulations on different topologies with random datasets demonstrate that the combination of the virtual machine assignment and the traffic engineering brings a significant improvement on the energy efficiency of the network in MapReduce systems. We prove that by using our framework, a further large amount of energy can be saved compared with the traffic engineering only solutions.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI*, 2004, pp. 137–150.
[2] U.s. environmental protection agency's data center report to congress. [Online]. Available: http://tinyurl.com/2jz3ft
[3] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," in *ISCA*, 2010, pp. 338–347.
[4] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: Saving energy in data center networks," in *NSDI*, 2010, pp. 249–264.
[5] Y. Shang, D. Li, and M. Xu, "Energy-aware routing in data center network," in *Green Networking*, 2010, pp. 1–8.
[6] P. Mahadevan, S. Banerjee, P. Sharma, A. Shah, and P. Ranganathan, "On energy efficiency for enterprise and data center networks," *IEEE Communications Magazine*, vol. 49, no. 8, pp. 94–100, 2011.
[7] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao, "Carpo: Correlation-aware power optimization in data center networks," in *INFOCOM*, 2012, pp. 1125–1133.
[8] N. Vasic, P. Bhurat, D. M. Novakovic, M. Canini, S. Shekhar, and D. Kostic, "Identifying and using energy-critical paths," in *CoNEXT*, 2011, p. 18.
[9] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint vm placement and routing for data center traffic engineering," in *INFOCOM*, 2012, pp. 2876–2880.
[10] L. Huang, Q. Jia, X. Wang, S. Yang, and B. Li, "Pcube: Improving power efficiency in data center networks," in *IEEE CLOUD*, 2011, pp. 65–72.
[11] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," in *WREN*, 2009, pp. 65–72.
[12] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *SIGCOMM*, 2008, pp. 63–74.
[13] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," in *SIGCOMM*, 2009, pp. 63–74.
[14] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," in *SIGCOMM*, 2008, pp. 75–86.
[15] M. Andrews, A. Fernández Anta, L. Zhang, and W. Zhao, "Routing for energy minimization in the speed scaling model," in *INFOCOM*, 2010, pp. 2435–2443.
[16] D. Xie, N. Ding, Y. C. Hu, and R. R. Kompella, "The only constant is change: incorporating time-varying network reservations in data centers," in *SIGCOMM*, 2012, pp. 199–210.
[17] T. C. Koopmans and M. Beckmann, "Assignment problems and the location of economic activities," *Econometrica*, vol. 25, no. 53-76, 1957.
[18] L. Wang, A. Fernández Anta, F. Zhang, C. Hou, and Z. Liu, "Energy-efficient network routing with discrete cost functions," in *TAMC*, 2012, pp. 307–318.
[19] Cplex. [Online]. Available: http://www.ilog.com/products/cplex